

ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN



Tiêu đề Bài tập nhóm

Nguyễn Hồng Phúc - 21000414
Nguyễn Minh An - 21002114

Mã học phần: MAT3508
Học kỳ 1, Năm học 2025-2026

Thông tin Dự án

Học phần: MAT3508 – Nhập môn Trí tuệ Nhân tạo
Học kỳ: Học kỳ 1, Năm học 2025-2026
Trường: VNU-HUS (Đại học Quốc gia Hà Nội – Trường Đại học Khoa học Tự nhiên)
Tên dự án: Using LLM in Movies Recommendation
Ngày nộp: 30/11/2025
Báo cáo PDF: Liên kết tới báo cáo PDF trong kho GitHub
Slide thuyết trình: Liên kết tới slide thuyết trình trong kho GitHub
Kho GitHub: <https://github.com/anbit192/AI-miniprj>

Thành viên nhóm

Họ tên	Mã sinh viên	Tên GitHub	Đóng góp
Nguyễn Hồng Phúc	21000414	titto2906	cách 2
Nguyễn Minh An	21002114	anbit192	cách 1

Mục lục

1	Giới thiệu	3
2	Công việc triển khai	4
2.1	Một vài công việc chung	4
2.2	Công việc đã làm	5
3	Review	6
4	Kiến thức nền tảng	8
4.1	Content-based Filtering	8
4.2	Collaborative Filtering	8
4.3	LLM Embedding	9
4.4	Vector Database	9
5	Phương pháp	11
5.1	Ý tưởng thuật toán	11
5.1.1	Content-based Filtering (LLM Embedding)	11
5.1.2	Hybrid Filtering (Collaborative + Content-based)	12
5.2	Kiến trúc hệ thống	14
5.2.1	Content-based Filtering (LLM Embedding)	14
5.2.2	Hybrid Filtering (Collaborative + Content-based)	14
6	Thực nghiệm	18
6.1	Dataset	18
6.2	Xử lý dữ liệu	19
6.3	Đánh giá	19
6.3.1	Content-based Filtering (LLM Embedding)	19
6.3.2	Hybrid Filtering	20
6.3.3	So sánh	20
7	Ứng dụng	21
7.1	User Interface	21
7.2	Chức năng	22
8	Kết luận	24

1 Giới thiệu

Trong kỷ nguyên số hóa hiện nay, ngành công nghiệp giải trí trực tuyến đang trải qua một cuộc cách mạng mạnh mẽ với sự bùng nổ của các nền tảng streaming như Netflix, Amazon Prime, Disney+, và nhiều dịch vụ khác. Tuy nhiên, cùng với sự phát triển này là thách thức lớn về "information overload" tình trạng người dùng bị choáng ngợp bởi hàng triệu bộ phim và chương trình TV có sẵn trên các nền tảng này.

Hệ thống gợi ý phim (Movie Recommendation System) đã trở thành một phần không thể thiếu trong việc giải quyết vấn đề này. Các nghiên cứu cho thấy rằng hơn 80% nội dung được xem trên Netflix đến từ các gợi ý của hệ thống, điều này không chỉ cải thiện trải nghiệm người dùng mà còn tăng đáng kể thời gian tương tác và doanh thu cho các nền tảng. Việc khai thác hiệu quả nguồn dữ liệu khổng lồ về sở thích, hành vi và đánh giá của người dùng để cung cấp những gợi ý cá nhân hóa, phù hợp với từng cá nhân, đang trở thành một bài toán trung tâm trong lĩnh vực trí tuệ nhân tạo và khai phá dữ liệu.

Bài toán được đặt ra trong nghiên cứu này là: "Làm thế nào để xây dựng một hệ thống gợi ý phim có khả năng hiểu sâu sắc về ngữ cảnh, sở thích cá nhân, và có thể tương tác tự nhiên với người dùng thông qua ngôn ngữ tự nhiên, đồng thời đảm bảo tính chính xác, đa dạng và khả năng mở rộng?". Các hệ thống gợi ý truyền thống như lọc cộng tác (collaborative filtering) và lọc theo nội dung (content-based filtering) thường bị giới hạn bởi khả năng xử lý ngôn ngữ nghĩa và khó thích ứng với các biểu đạt ngôn ngữ tự nhiên đa dạng của người dùng. Collaborative filtering gặp vấn đề cold start và sparsity, trong khi content-based filtering dễ rơi vào filter bubble và khó mở rộng cho các đặc trưng phức tạp. Trong bối cảnh đó, việc ứng dụng các mô hình ngôn ngữ lớn (LLM) nổi lên như một hướng tiếp cận đầy triển vọng, nhờ vào khả năng hiểu và sinh ngôn ngữ tự nhiên một cách linh hoạt, sắc nét.

Việc áp dụng các mô hình ngôn ngữ lớn (LLM) – mô hình học sâu tiên tiến được huấn luyện trên khối lượng dữ liệu văn bản khổng lồ – vào hệ thống gợi ý phim mở ra một hướng đi mới đầy hứa hẹn. Với khả năng hiểu ngữ cảnh tự nhiên và xử lý ngôn ngữ tinh vi, LLM cho phép phân tích cảm xúc, sở thích cá nhân từ phản hồi người dùng, đồng thời tạo ra gợi ý phim phù hợp dựa trên nội dung và đặc trưng của từng bộ phim. Hệ thống giúp người dùng khám phá các tác phẩm không chỉ dựa trên thể loại hay điểm đánh giá mà còn phù hợp với tâm trạng, sở thích và bối cảnh cá nhân.

Dự án này tập trung xây dựng một hệ thống gợi ý phim dựa trên sự kết hợp giữa mô hình ngôn ngữ lớn Gemini, cơ sở dữ liệu vector FAISS, FastAPI cho backend, MongoDB cho cơ sở dữ liệu phi quan hệ, và Streamlit cho giao diện người dùng. Mỗi thành phần được lựa chọn đều nhằm tối ưu hóa từng khía cạnh trong quá trình xây dựng, triển khai và vận hành hệ thống. Cụ thể về công nghệ, công cụ được sử dụng:

- Google Gemini (LLM): Gemini được thiết kế với kiến trúc multimodal tiên tiến, có khả năng xử lý đồng thời text, image, và video - điều cực kỳ quan trọng cho domain phim ảnh. Mô hình này được tối ưu hóa cho các tác vụ reasoning phức tạp và có khả năng context understanding vượt trội, cho phép hiểu sâu ý định của người dùng qua các truy vấn tự nhiên và phân tích nội dung phim một cách tinh tế.
- FAISS (Facebook AI Similarity Search): Được thiết kế tối ưu cho tìm kiếm tương đồng trên không gian vector lớn, FAISS cho phép lưu trữ và truy vấn các biểu diễn vector của phim với tốc độ cao, độ chính xác tốt, và khả năng mở rộng mạnh mẽ. FAISS nổi bật với khả năng indexing hiệu quả, hỗ trợ GPU acceleration, và có thể xử lý hàng tỷ vectors với tốc độ millisecond.

- FastAPI (Backend): FastAPI là một framework hiện đại, nhẹ và hiệu quả cho phát triển API bằng Python. Được xây dựng trên Starlette và Pydantic, FastAPI cung cấp automatic API documentation, type hints support, và async/await capabilities mạnh mẽ. Benchmark tests cho thấy FastAPI có performance gần tương đương với NodeJS và nhanh hơn đáng kể so với Django hay Flask.
- MongoDB (Database): MongoDB là hệ quản trị cơ sở dữ liệu NoSQL phù hợp lưu trữ dữ liệu phi cấu trúc như thông tin phim, người dùng, lịch sử tương tác. Movie metadata thường có cấu trúc semi-structured với nhiều nested fields như cast, genres, production companies, và user-generated content. MongoDB's document model phù hợp hoàn hảo cho việc lưu trữ và query các dữ liệu này.
- Streamlit (Frontend): Streamlit cho phép xây dựng giao diện người dùng trực quan, dễ phát triển, tối ưu cho các ứng dụng AI/ML. Mặc dù không mạnh bằng React hay Vue.js về customization, Streamlit cho phép data scientists và ML engineers xây dựng interactive web applications mà không cần kiến thức sâu về web development.

Mỗi công nghệ được lựa chọn đều có những ưu điểm vượt trội phù hợp với yêu cầu của hệ thống. Gemini mang lại khả năng hiểu ngữ nghĩa sâu sắc và tổng quát hóa vượt trội, cung cấp api miễn phí, tuy nhiên có giới hạn request mỗi người dùng. FAISS cung cấp tốc độ tìm kiếm vector nhanh chóng nhưng chủ yếu phù hợp với dữ liệu dạng vector. FastAPI đảm bảo hiệu suất cao và dễ phát triển nhưng cần hiểu về async programming. MongoDB cung cấp tính linh hoạt cao với dữ liệu phi cấu trúc nhưng không bảo đảm toàn vẹn như hệ thống quan hệ. Streamlit cho phép phát triển frontend nhanh chóng nhưng không thể thiết kế giao diện một cách chi tiết.

Sự lựa chọn này được đưa ra dựa trên sự cân bằng giữa hiệu năng, tốc độ phát triển, tính linh hoạt và khả năng mở rộng. Đặc biệt, Gemini giúp nâng cao năng lực hiểu ngôn ngữ tự nhiên của hệ thống, FAISS đảm trách phần tìm kiếm tương đồng theo vector hiệu quả, trong khi FastAPI và MongoDB tạo thành một nền tảng backend linh hoạt, dễ mở rộng. Streamlit giúp nhanh chóng hiện thực hóa giao diện người dùng, tạo điều kiện thuận lợi cho thử nghiệm và cải tiến sản phẩm.

Sự kết hợp các công nghệ trên không chỉ giúp xây dựng một hệ thống gợi ý phim hiện đại, hiệu quả, mà còn mở ra tiềm năng mở rộng cho nhiều lĩnh vực ứng dụng khác. Đặc biệt, khả năng hiểu và tương tác ngôn ngữ tự nhiên nhờ sức mạnh của LLM sẽ là chìa khóa để nâng tầm trải nghiệm cá nhân hóa, đưa hệ thống gợi ý lên một tầm cao mới – nơi công nghệ không chỉ hỗ trợ mà còn “đồng hành” cùng người dùng trong hành trình khám phá nghệ thuật thứ bảy.

2 Công việc triển khai

2.1 Một vài công việc chung

- Xử lý, tổ chức, tổng hợp dữ liệu và càò thêm dữ liệu từ API để làm phong phú thêm dữ liệu.
- Tìm hiểu về các LLM: LLama và Gemini.
- Tìm hiểu về quy trình RAG để ứng dụng vào bài toán.
- Tìm hiểu về các phương pháp ứng dụng LLM vào hệ thống recommend.
- CSS cho front-end.

2.2 Công việc đã làm

Sinh viên Nguyễn Minh An

- Phát triển hệ thống gợi ý content-based với LLM embeds làm đầu vào:
 - Áp dụng khôi embed của LLM để tạo vector database cho dữ liệu phim.
 - Phân tách logic hệ thống thành các file và lớp riêng.
 - Phát triển và tối ưu luồng chạy hệ thống.
 - Áp dụng đánh giá score ranking và hệ thống thưởng phạt.
 - Xây dựng phương pháp đánh giá hệ thống.
- Xây dựng và phát triển ứng dụng web:
 - Phát triển API.
 - Tích hợp với MongoDB để quản lý dữ liệu user.

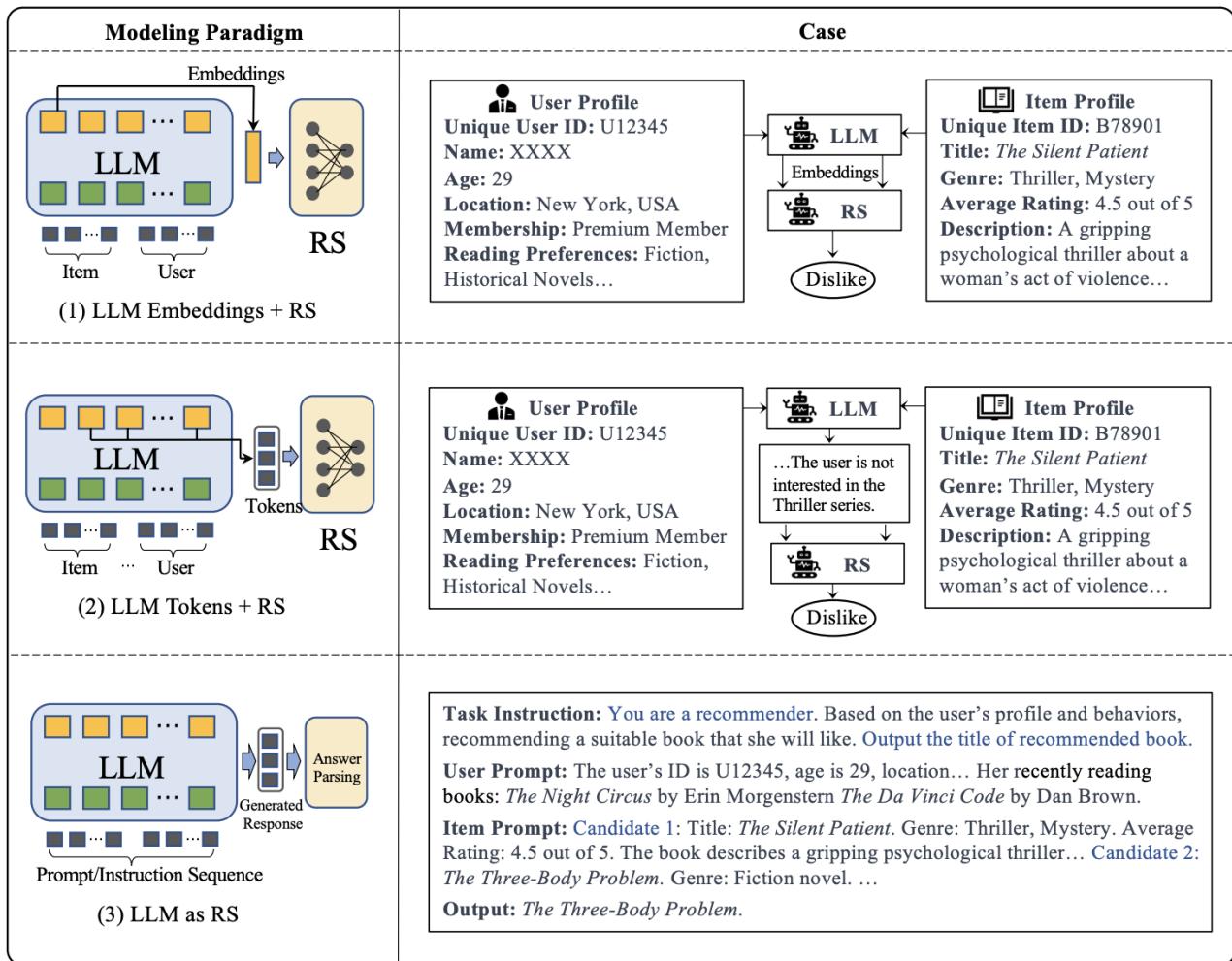
Sinh viên Nguyễn Hồng Phúc

- Phát triển hệ thống gợi ý Hybrid ứng dụng LLM:
 - Kết hợp các phương pháp truyền thống để tăng hiệu suất mô hình.
 - Xây dựng phương pháp content-based mới tận dụng sức mạnh LLM.
 - Phát triển và tối ưu phương pháp mới.
 - Đánh giá so sánh với các phương pháp khác nhau.
- Xây dựng và phát triển ứng dụng web:
 - Phát triển UI.

3 Review

Với sự thành công của các mô hình LLM (BERT, GPT, PaLM, LLaMA, ...) trong xử lý ngôn ngữ tự nhiên, việc ứng dụng chúng vào hệ thống gợi ý đã mở ra hướng đi mới để khắc phục hạn chế của phương pháp truyền thống. Nhiều nghiên cứu đã được thực hiện xoay quanh chủ đề này.

Tác giả của A Survey on Large Language Models for Recommendation (2024) [2] đã phân LLM thành 3 vai trò chính trong hệ thống gợi ý:



- LLM Embeddings + RS:** LLM được sử dụng như một bộ trích xuất đặc trưng, chuyển đổi thông tin người dùng và mô tả sản phẩm thành các vector (embeddings) sau đó đưa vào mô hình gợi ý truyền thống. Ưu điểm của cách làm này là nó khai thác triệt để kiến thức phong phú được học qua quá trình huấn luyện tự giám sát của LLM, giúp khắc phục các vấn đề về dữ liệu thưa và cold-start, đồng thời cải thiện độ chính xác của hệ thống.
- LLM Tokens + RS:** Thay vì chỉ sử dụng vector, LLM có thể chuyển đổi dữ liệu thành chuỗi token ngôn ngữ tự nhiên, lưu giữ thông tin ngữ cảnh và diễn giải chi tiết, hỗ trợ quá trình gợi ý.
- LLM as RS:** LLM được tích hợp trực tiếp để sinh ra các đề xuất gợi ý dưới dạng văn bản, sử dụng khả năng sinh văn bản của mô hình thông qua các kỹ thuật như prompt tuning hoặc instruction tuning, phù hợp cho các ứng dụng hội thoại.

Còn theo Large Language Model Enhanced Recommender Systems: A Survey(2025) [1], có 3

phương pháp chính để tích hợp LLM vào hệ thống:

1. Tăng cường kiến thức (Knowledge Enhancement):

Sử dụng các mô hình ngôn ngữ lớn (LLM) để tạo ra kiến thức có cấu trúc hoặc không có cấu trúc (như tóm tắt văn bản hoặc đồ thị tri thức) nhằm làm phong phú thêm đặc trưng của người dùng hoặc sản phẩm.

2. Tăng cường tương tác (Interaction Enhancement):

Giúp giảm thiểu vấn đề dữ liệu thưa thớt bằng cách sử dụng LLM để tổng hợp các tương tác giữa người dùng và sản phẩm nhằm phục vụ cho quá trình huấn luyện mô hình.

3. Tăng cường mô hình (Model Enhancement):

Tận dụng embedding do LLM tạo ra hoặc kiến thức đã được chưng cất từ LLM để cải thiện các mô hình gợi ý truyền thống mà không cần sử dụng LLM trong quá trình suy luận (inference).

4 Kiến thức nền tảng

4.1 Content-based Filtering

Định nghĩa: Content-based filtering (lọc dựa trên nội dung) là phương pháp xây dựng hồ sơ người dùng dựa trên đặc trưng của các mục (items) mà người dùng đã thích hoặc tương tác. Hệ thống sau đó sử dụng các đặc trưng này để gợi ý các mục có nội dung tương tự mà người dùng có thể quan tâm.

Cách thức hoạt động:

Phân tích nội dung: Hệ thống xác định các đặc trưng quan trọng của các mục, có thể là từ khóa, thể loại, thuộc tính kỹ thuật, hoặc các thuộc tính mô tả khác. Ví dụ, trong một hệ thống gợi ý phim, đặc trưng có thể bao gồm thể loại (hành động, hài, tâm lý, ...), diễn viên, đạo diễn, v.v.

Xây dựng hồ sơ người dùng: Hồ sơ của người dùng được tạo ra dựa trên các mục mà họ đã tương tác (đánh giá, mua, xem, ...). Hồ sơ này phản ánh sở thích cá nhân của người dùng dựa trên đặc trưng của các mục đó.

So sánh và gợi ý: Hệ thống so sánh hồ sơ người dùng với các mục trong cơ sở dữ liệu và gợi ý những mục có đặc trưng phù hợp với sở thích người dùng.

Ưu điểm:

- Có thể hiểu rõ sở thích riêng biệt của từng người dùng.
- Đề xuất các mục mới mà chưa có ý kiến từ người dùng khác, miễn là có thông tin mô tả đầy đủ.

Nhược điểm:

- Khó khăn trong việc mô tả đầy đủ các thuộc tính của mục (đặc biệt là với các mục mang tính chủ quan như nghệ thuật).
- Khả năng "over-specialization": hệ thống chỉ gợi ý các mục rất giống với những mục đã được tương tác trước đó, không đa dạng hóa đề xuất.
- Không khắc phục được vấn đề "cold start" đối với người dùng mới nếu không có đủ dữ liệu về sở thích của họ.

4.2 Collaborative Filtering

Định nghĩa: Collaborative filtering (lọc cộng tác) dựa trên hành vi và sở thích của nhiều người dùng. Phương pháp này giả định rằng nếu hai người dùng có sở thích tương tự trong quá khứ, thì họ có khả năng có cùng sở thích trong tương lai.

Cách thức hoạt động:

- User-based Collaborative Filtering:

Tìm kiếm những người dùng khác có hành vi tương tự (ví dụ: đánh giá, xem, mua hàng) với người dùng hiện tại.

Gợi ý những mục mà các người dùng tương tự đã thích nhưng người dùng hiện tại chưa từng tương tác.

- Item-based Collaborative Filtering:

Dựa trên mối liên hệ giữa các mục: nếu một người dùng thích mục A, và nhiều người

dùng khác có xu hướng thích cùng lúc mục A và mục B, thì hệ thống sẽ gợi ý mục B cho người dùng đó.

Ưu điểm:

- Không cần phải phân tích nội dung chi tiết của từng mục.
- Tận dụng sức mạnh của dữ liệu cộng đồng, do đó có thể phát hiện ra các sở thích mà có thể không rõ ràng qua mô tả nội dung.
- Có khả năng đề xuất các mục mới, thậm chí nếu không có thông tin chi tiết về chúng, miễn là có dữ liệu tương tác từ cộng đồng.

Nhược điểm:

- Vấn đề cold start: Nếu người dùng mới hoặc mục mới chưa có đủ dữ liệu tương tác, việc gợi ý sẽ gặp khó khăn.
- Sự thừa thót ("sparsity"): Trong các hệ thống lớn với số lượng mục và người dùng khổng lồ, ma trận tương tác thường rất thừa, dẫn đến việc tìm kiếm mỗi liên hệ chính xác giữa các mục hoặc người dùng có thể gặp khó khăn.
- Scalability: Khi số lượng người dùng và mục tăng lên rất nhiều, tính toán và duy trì các mô hình tương tác có thể đòi hỏi tài nguyên tính toán lớn.

4.3 LLM Embedding

Trong các mô hình ngôn ngữ lớn (LLM), khối *Embedding* chịu trách nhiệm chuyển đổi văn bản đầu vào (chuỗi ký tự, từ, hay các token) thành các vector số học. Các vector này biểu diễn đầy đủ thông tin ngữ nghĩa và cấu trúc của dữ liệu, giúp hệ thống có khả năng xử lý, so sánh và hiểu mối quan hệ giữa các từ. Các bước chính bao gồm:

- **Tokenization:** Phân tách văn bản thành các đơn vị cơ bản (tokens) như từ hoặc các phần của từ.
- **Mapping (Ánh xạ):** Mỗi token được ánh xạ sang một vector trong không gian đa chiều với kích thước cố định, sao cho các token có ý nghĩa tương tự có vector gần nhau.
- **Positional Encoding:** Do kiến trúc Transformer không lưu giữ thông tin thứ tự, nên một mã hoá vị trí được thêm vào mỗi vector để biểu hiện vị trí của token trong chuỗi.

Ý Nghĩa của Embedding: Sau quá trình embedding, văn bản được chuyển thành dạng số học (chuỗi các vector), tạo nền tảng cho các lớp tiếp theo của mô hình (như các lớp Transformer) trong việc xử lý thông tin dựa trên các phép toán ma trận và cơ chế tự chú ý (self-attention).

4.4 Vector Database

Vector Database là một loại cơ sở dữ liệu được tối ưu hóa để lưu trữ, truy vấn và tìm kiếm các vector – thường là các embedding được tạo ra từ mô hình học máy, đặc biệt là trong các ứng dụng liên quan đến xử lý ngôn ngữ tự nhiên, xử lý hình ảnh hay các hệ thống gợi ý.

Định Nghĩa và Vai Trò

- **Vector Embedding:** Trong các ứng dụng AI, dữ liệu như văn bản, hình ảnh, âm thanh,... thường được chuyển đổi thành các vector (dãy số). Các vector này biểu diễn đặc tính ngữ nghĩa của dữ liệu, cho phép đánh giá mức độ tương đồng giữa các mục.

- **Mục Đích:** Vector Database được thiết kế để lưu trữ và truy xuất hàng triệu, thậm chí hàng tỷ vector, nhằm đáp ứng yêu cầu tìm kiếm "nearest neighbors" (các vector gần nhất) một cách nhanh chóng và hiệu quả.
- **Ứng Dụng:** Các hệ thống tìm kiếm ngữ nghĩa, gợi ý sản phẩm, nhận dạng hình ảnh,... sử dụng vector database để truy vấn các vector embedding dựa trên độ tương đồng.

Các Thành Phần Chính của Vector Database

- **Lưu Trữ Vector:** Khả năng lưu trữ một khối lượng lớn vector và truy xuất nhanh chóng.
- **Indexing (Chỉ mục hóa):** Sử dụng các cấu trúc dữ liệu và thuật toán (như KD-Tree, HNSW, IVF,...) để tăng tốc truy vấn, đảm bảo việc tìm "nearest neighbor" hiệu quả ngay cả với tập dữ liệu lớn.
- **Truy Vấn Tương Đồng:** Hỗ trợ các phép đo khoảng cách (ví dụ: Euclid, cosine similarity) để so sánh các vector và xác định mức độ tương đồng.

5 Phương pháp

5.1 Ý tưởng thuật toán

5.1.1 Content-based Filtering (LLM Embedding)

Chuyển thông tin phim thành chuỗi lớn, ví dụ:

title: ... | genres: ...

Tận dụng khả năng embed của LLM để vector hóa những chuỗi đó, sau đó lưu những vector đó vào vector database. Ngoài việc tìm những phim giống, ta còn phải quan tâm tới phim chủ thể tìm kiếm được user đánh giá cao hay thấp (thích hay không thích). Nhóm đã đặt ra ngưỡng là 3.0 sao. Do đó ta cần một hệ thống đánh giá ranking của phim, “thưởng” cho những phim giống với phim mà user thích (rate ≥ 3) và “phạt” những phim mà user ko thích (rate < 3).

Mỗi phim được tính giá trị *weight rating* theo công thức Bayesian average:

$$\text{weight rating} = \frac{v \cdot R + m \cdot C}{v + m}$$

trong đó:

- R : điểm trung bình của phim.
- v : số lượt đánh giá của phim.
- m : số lượt đánh giá tối thiểu.
- C : điểm trung bình toàn bộ phim.

Giá trị weight rating có thể coi là đánh giá tổng thể của phim, công thức này giúp cân bằng giữa phim có nhiều lượt xem và phim có ít lượt xem.

Sau đó ta kết hợp đánh giá riêng của user với phim và weight rating của phim đó, ta có thể tùy chỉnh trọng số cho cả 2 loại đánh giá, ở đây nhóm chọn trọng số cho đánh giá riêng ở mức 0.8 và trọng số cho weight rating ở mức 0.2:

$$\text{Combine rating} = 0.8 \cdot \text{user rate} + 0.2 \cdot \text{weight rating}$$

Trọng số cho user rate cao để ưu tiên việc user thích hay ko thích phim hơn là phim đó có nổi tiếng hay không

Từ lịch sử xem phim (Có thể lấy 3-4 phim đã xem gần đây và 1-2 phim đã xem từ trước) của user, ứng với mỗi phim thứ i , ta lấy ra n phim giống với phim thứ i đó:

- Nếu user thích phim đó thì những phim j (trong n phim) giống với phim thứ i đó sẽ được “thưởng” theo công thức:

$$\text{Score}(j) += \text{combine rating} \times \text{similarity}(j, i)$$

- Nếu user không thích phim đó thì những phim j (trong n phim) giống với phim thứ i đó sẽ bị “phạt” theo công thức:

$$\text{Score}(j) += \left(\text{combine rating} - \alpha \cdot (5 - \text{user rate}(i)) \right) \times \text{similarity}(j, i)$$

α càng lớn thì mức phạt càng cao, nhóm chọn $\alpha = 1$

Sau khi duyệt hết lịch sử xem của user, ta sẽ thu được danh sách $n * \text{len(watched movies)}$ phim ứng với điểm ranking của chúng, từ đó ta sắp xếp các phim theo thứ tự điểm ranking giảm dần và lấy ra top - k làm kết quả cuối cùng của hệ thống.

5.1.2 Hybrid Filtering (Collaborative + Content-based)

Hybrid Filtering là sự kết hợp của hai hoặc nhiều phương pháp nhằm tận dụng điểm mạnh và khắc phục điểm yếu của từng phương pháp riêng lẻ. Bằng cách tích hợp cả dữ liệu người dùng (user-item interactions) và thông tin nội dung (content features), hệ thống có thể cung cấp các gợi ý chính xác và cá nhân hóa hơn, đồng thời giảm thiểu các vấn đề như cold-start hoặc data sparsity. Trong dự án này, nhóm sử dụng mô hình Hybrid Filtering kết hợp giữa Collaborative Filtering (lọc cộng tác) và Content-based Filtering (lọc theo nội dung).

Collaborative Filtering (CF)

Collaborative Filtering là phương pháp truyền thống trong hệ thống gợi ý, dựa vào hành vi và sở thích của người dùng thay vì thông tin nội dung của sản phẩm. Ý tưởng chính là: "Một người dùng có thói quen thích một loại sản phẩm trong quá khứ thì họ sẽ thích những sản phẩm tương tự trong tương lai."

Collaborative Filtering chia thành hai loại chính:

- Memory-based: Sử dụng trực tiếp ma trận người dùng – sản phẩm để tính toán sự tương đồng và đưa ra dự đoán.
- Model-based: Sử dụng kỹ thuật học máy để học các đặc trưng ẩn (latent features) từ dữ liệu đánh giá.

Nhóm sử dụng hướng tiếp cận Model-based với model là Singular Value Decomposition (SVD)

$$R \approx U \cdot \Sigma \cdot V^T$$

trong đó:

- $R \in \mathbb{R}^{m \times n}$: ma trận đánh giá người dùng–phim.
- $U \in \mathbb{R}^{m \times k}$: ma trận đặc trưng người dùng.
- $V \in \mathbb{R}^{n \times k}$: ma trận đặc trưng phim.
- k : số chiều đặc trưng ẩn.

Dự đoán đánh giá của người dùng u cho phim i được tính theo:

$$\hat{r}_{ui} = p_u \cdot q_i$$

- p_u : vector đặc trưng ẩn của người dùng u
- q_i : vector đặc trưng ẩn của item i

Với model SVD cho Collaborative Filtering, hệ thống có thể hoạt động hiệu quả với dữ liệu lớn; khai quát, khai thác tốt đặc trưng ẩn và có thể dự đoán cả những mục chưa từng được đánh giá trực tiếp nhờ latent factors. Tuy nhiên, cold-start luôn là vấn đề lớn với phương pháp này, do phương pháp dựa vào điểm đánh giá nên những bộ phim hoặc người dùng mới sẽ thiếu dữ liệu điểm đánh giá, ảnh hưởng đến kết quả.

Tích hợp Hybrid Filtering

Để tận dụng ưu điểm của cả hai phương pháp Collaborative Filtering (CF) và Content-based

Filtering (CBF), Nhóm xây dựng một mô hình Hybrid Filtering bằng cách kết hợp các điểm số đề xuất phim từ cả hai phương pháp, đồng thời bổ sung thêm yếu tố Movie Score – phản ánh mức độ nổi bật và đánh giá tổng thể của bộ phim. Điều này vừa khắc phục được cold-start của CF và cũng giải quyết được vấn đề sparsity của bộ dữ liệu.

Với đầu vào là một người dùng với những bộ phim đã xem cùng điểm số của nó. Hệ thống sẽ tính điểm hybrid cho tất cả những bộ phim mà họ chưa xem với công thức:

$$\text{Hybrid Score} = 0.5 \cdot \text{CF Score} + 0.3 \cdot \text{Semantic Score} + 0.2 \cdot \text{Movie Score}$$

- CF Score: Điểm dự đoán từ mô hình SVD Collaborative Filtering, đã được chuẩn hóa về khoảng [0,1]. Phản ánh độ phù hợp của phim dựa trên hành vi cộng đồng.
- Semantic score: Điểm tương đồng ngữ nghĩa (cosine similarity) giữa user summary và movie summary trong phương pháp Content-based Filtering. Phản ánh sự tương đồng về mặt nội dung và sở thích cá nhân.
- Movie Score: Điểm tổng hợp đánh giá chất lượng phim.

Hệ thống sẽ lấy ra top k bộ phim với số điểm Hybrid score cao nhất. **Cách tính điểm Movie Score**

$$\text{Movie Score} = 0.4 \cdot \text{Weighted Average} + 0.6 \cdot \text{Popularity Score}$$

Popularity Score được tính dựa trên số lượng lượt đánh giá (votes) mà bộ phim nhận được. Phim càng có nhiều lượt đánh giá thì càng được coi là phổ biến. Để chuẩn hóa giá trị này về khoảng [0,1], Nhóm sử dụng công thức min-max normalization:

$$\text{Popularity}_{\text{norm}} = \frac{v - v_{\min}}{v_{\max} - v_{\min}}$$

- v : số lượng lượt đánh giá của phim
- v_{\min}, v_{\max} : số lượt đánh giá nhỏ nhất và lớn nhất trong toàn bộ tập dữ liệu

Weighted average là phương pháp đánh giá phim có xét đến chất lượng (rating) và số lượng đánh giá (votes), tránh trường hợp các phim có ít lượt đánh giá nhưng lại có điểm trung bình cao gây sai lệch. Đây là cách đánh giá uy tín được trang web đánh giá phim nổi tiếng IMDB sử dụng. Công thức đã được đề cập ở phương pháp đầu.

Mục tiêu của **Movie Score** là đảm bảo rằng phim có số lượng đánh giá lớn sẽ ảnh hưởng mạnh đến điểm, trong khi phim có ít đánh giá sẽ bị “kéo về” gần điểm trung bình của toàn bộ dữ liệu, nhằm tạo ra một đánh giá công bằng. Tỷ lệ trọng số 40% và 60% là để thực hiện điều ấy, tức chúng ta ưu tiên những bộ phim hot cho dù đánh giá của nó tệ hay không.

Ưu điểm:

- Cân bằng giữa cá nhân hóa và xu hướng cộng đồng.
- Giảm tác động của *cold-start*, vì phương pháp *Content-based* và *Movie Score* vẫn hoạt động tốt với người dùng hoặc phim mới.
- Tăng độ chính xác và độ tin cậy của gợi ý nhờ sử dụng thông tin ngữ nghĩa chuyên sâu (*semantic information*).

Nhược điểm:

- Việc sinh tóm tắt cho phim và người dùng tốn nhiều thời gian và tài nguyên khi dữ liệu càng nhiều.

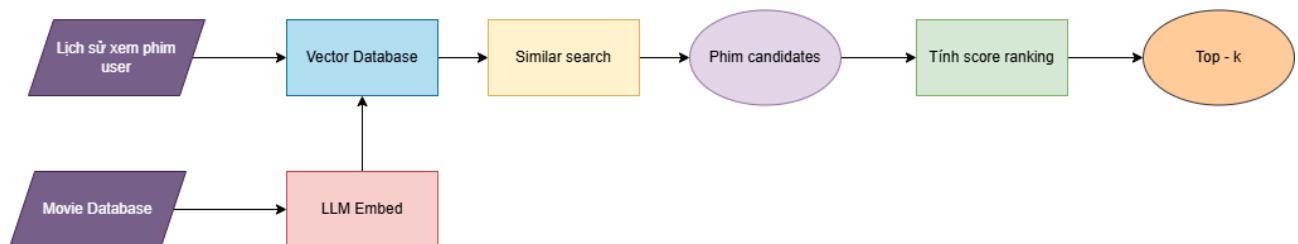
Ý nghĩa:

- Tận dụng khá triệt để sức mạnh của LLM: khả năng khái quát, hiểu ngữ nghĩa và cá nhân hóa.
- Kết quả gợi ý dựa trên đa dạng tiêu chí:
 - Thị hiếu cộng đồng
 - Chất lượng bộ phim
 - Phù hợp với sở thích người dùng
 - Độ nổi tiếng

5.2 Kiến trúc hệ thống

- **Vector Database:** FAISS.
- **Database:** MongoDB cục bộ.
- **LLM Embed Model:** text-embed-004 (Gemini Flash 2.0).
- **LLM Generative Model:** Gemini Flash 2.0.
- **Backend:** FastAPI.

5.2.1 Content-based Filtering (LLM Embedding)



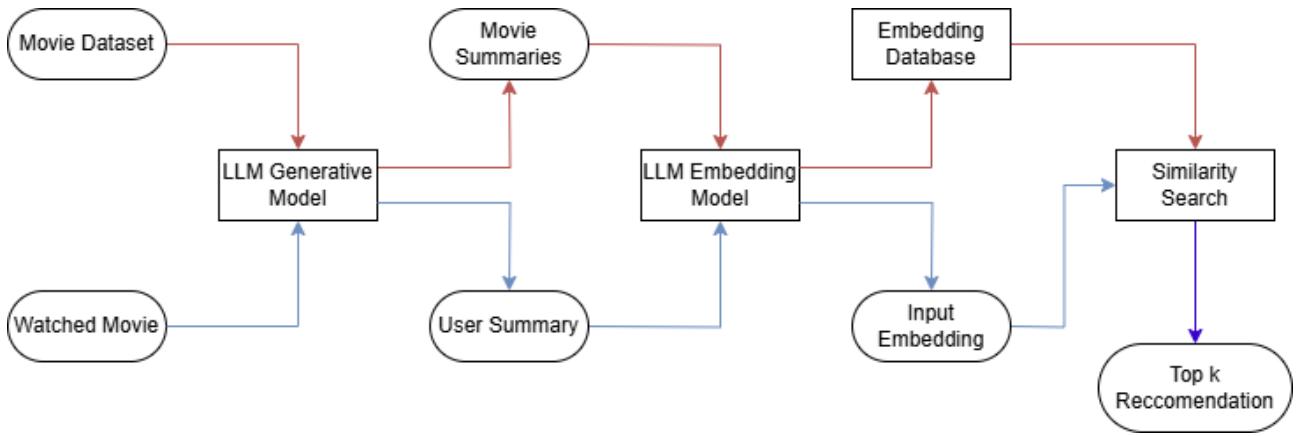
Hình 1: Content-based với LLM embed

Luồng hoạt động:

1. Load dữ liệu khi server khởi động.
2. Lấy lịch sử xem của người dùng.
3. Chuyển dữ liệu lịch sử xem sang định dạng phù hợp cho hệ thống gợi ý.
4. Phân loại kết quả gợi ý thành các thể loại.

5.2.2 Hybrid Filtering (Collaborative + Content-based)

Text Summary (Content-based):



Hình 2: Text Summary (Content-based)

1 - Sinh tóm tắt phim (Movie Summaries)

- Input:** Tập dữ liệu phim (Movie Dataset) chứa các thông tin tiêu đề, thể loại, tags.
- Processing:** Dữ liệu này được đưa vào mô hình **LLM Generative Model** thông qua *prompt*:

```

"Analyze these movies and summarize each in this EXACT
format:
Genres: [genres] | Themes: [3-5 themes] | Style: [2-3 style
words] | Notable: [1-3 standout elements]""

Input Data:
<Movie Data>

Provide exactly <number of movie> summaries, one per line:"

```

Hình 3: Promt sinh tóm tắt phim

- Output:** Mỗi phim có một 1 câu **Movie Summary** dạng văn bản tự nhiên mô tả thể loại, chủ đề, phong cách và yếu tố nổi bật của phim.

2 - Sinh tóm tắt người dùng (User Summary)

- Input:** 100 bộ phim đã xem gần nhất của người dùng (Watched Movies)
- Processing:** Các movie summaries tương ứng với các phim đã xem được gom lại và đưa vào **LLM Generative Model** thông qua *prompt*:

“Analyze the following movie ratings and summaries to infer
 this user's preferences EXACTLY this format:
 Genres: [5-6 favorite genres] | Themes: [4-5 favorite
 themes] | Style: [3-4 stylistic preferences] | Notable: [2-3
 standout elements they seem to enjoy]

Rated Movies:
 <Watched Movies Summaries>

 Based on the above, write a one-line summary of the user's
 preferences.”

Hình 4: prompt sinh tóm tắt người dùng

- **Output:** Một câu văn bản tổng hợp ngữ nghĩa mô tả thể loại, chủ đề, phong cách và yếu tố nổi bật mà người dùng yêu thích.

3 - Chuyển văn bản thành vector embedding

- **Movie Summaries** và **User Summary** được đưa vào **LLM Embedding Model** để chuyển thành các vector số có ý nghĩa ngữ nghĩa.
- **Kết quả:**
 - Movie Embeddings được lưu trữ trong **Embedding Database**.
 - User Summary Embedding tạo thành **Input Embedding** cho truy vấn tìm kiếm.

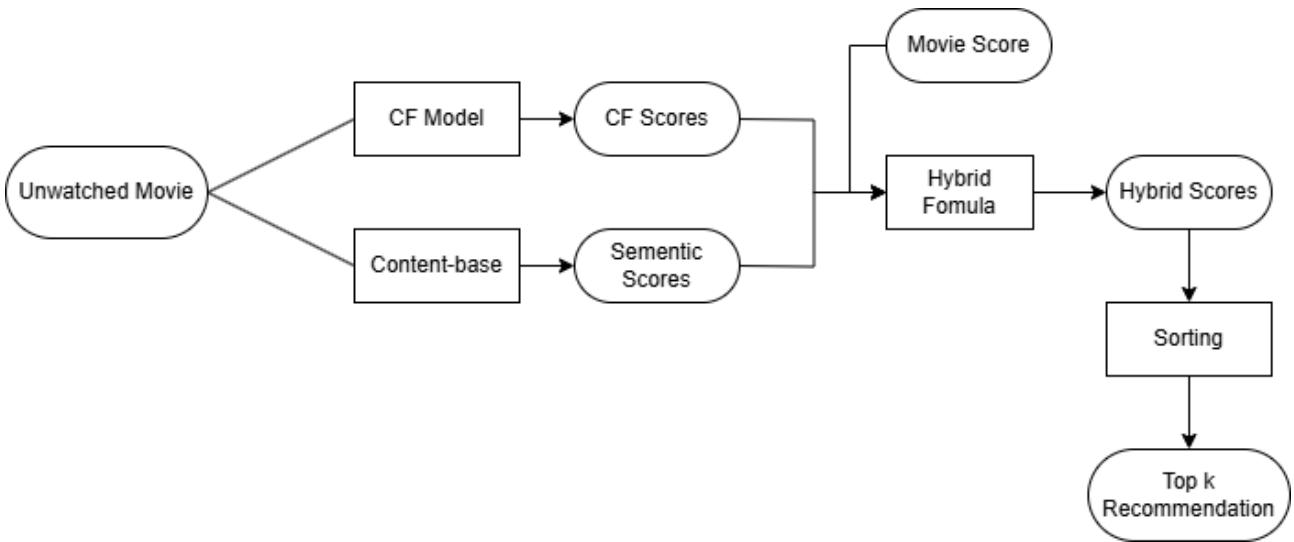
4 - Tìm kiếm phim tương đồng (Similarity Search)

- Sử dụng **Input Embedding** (biểu diễn sở thích người dùng) để tìm kiếm trong **Embedding Database** các vector phim có độ tương đồng cosine cao nhất.
- **Similarity Search** thực hiện phép so sánh cosine và trả về **Top-k Recommendation** (gọi ý k bộ phim có điểm số cosine cao nhất).

Hybrid Filtering

Hệ thống gợi ý kết hợp (Hybrid Filtering) được xây dựng dựa trên việc kết hợp điểm số từ 3 nguồn:

- Collaborative Filtering (CF) – mô hình SVD
- Content-based Filtering – dựa trên semantic similarity giữa user summary và movie summary
- Movie Score – đánh giá chất lượng và mức độ phổ biến của phim



Hình 5: Hybrid Filtering

1 - Gợi ý từ Content-based Filtering (Semantic Similarity)

- Input:** Dữ liệu phim + lịch sử xem của người dùng
- Output:** Một danh sách phim chưa xem kèm theo điểm semantic score (0 - 1)

2 - Gợi ý từ Collaborative Filtering (SVD)

- Input:** Ma trận người dùng – phim với rating gốc đã được chuẩn hóa về 0 - 1
- Xử lý:** Dùng mô hình SVD để dự đoán điểm rating cho tất cả các phim mà người dùng chưa xem
- Output:** Danh sách phim với điểm CF score (0 - 1).

3 - Tính điểm Movie Score

- Popularity Score:** Dựa trên số lượt đánh giá (đã chuẩn hóa)
- Weighted Average:** Tính theo công thức IMDb
- Movie Score:**

$$\text{Movie Score} = 0.4 \times \text{Weighted Average} + 0.6 \times \text{Popularity Score}$$

4 - Sinh gợi ý

Với mỗi phim, tính điểm gợi ý tổng hợp theo công thức:

$$\text{Hybrid Score} = 0.5 \times \text{CF Score} + 0.3 \times \text{Semantic Score} + 0.2 \times \text{Movie Score}$$

Kết quả: Danh sách phim được sắp xếp theo Hybrid Score, sau đó chọn ra Top-k phim gợi ý cuối cùng cho người dùng.

6 Thực nghiệm

6.1 Dataset

Sử dụng bộ dữ liệu MovieLens 20M với các thông số chính:

- **Số lượng đánh giá:** 20.000.263 (thang điểm 0.5–5.0).
- **Số lượng gắn thẻ:** 465.564.
- **Số lượng phim:** 27.278.
- **Số lượng người dùng:** 138.493.
- **Thời gian thu thập:** từ 9/1/1995 đến 31/3/2015.

Các tệp dữ liệu chính bao gồm: **ratings.csv**, **movies.csv**, **tags.csv**, **links.csv**, **genome-scores.csv**, **genome-tags.csv**. Ngoài ra còn kết hợp thêm dữ liệu từ IMDB (OMDB API)

1. **ratings.csv:** Thông tin đánh giá phim của người dùng

- **userId:** ID của người dùng
- **movieId:** ID của phim
- **rating:** Điểm đánh giá (0.5 đến 5.0)
- **timestamp:** Thời điểm đánh giá (UNIX timestamp)

2. **movies.csv:** Thông tin chi tiết về phim

- **movieId:** ID của phim
- **title:** Tiêu đề phim (bao gồm năm phát hành)
- **genres:** Thể loại phim (phân tách bằng dấu |)

3. **tags.csv:** Thẻ (*tags*) do người dùng gắn cho phim

- **userId:** ID của người dùng
- **movieId:** ID của phim
- **tag:** Tên thẻ
- **timestamp:** Thời điểm gắn thẻ

4. **links.csv:** Liên kết phim đến các ID trong IMDb và TMDb

- **movieId:** ID của phim trong MovieLens
- **imdbId:** ID trong IMDb
- **tmdbId:** ID trong TMDb

5. **genome-scores.csv:** Điểm liên quan giữa phim và thẻ trong bộ “tag genome”

- **movieId:** ID của phim
- **tagId:** ID của thẻ
- **relevance:** Mức độ liên quan (giá trị từ 0 đến 1)

6. **genome-tags.csv:** Danh sách thẻ trong bộ “tag genome”

- `tagId`: ID của thẻ
- `tag`: Tên thẻ

Dữ liệu bổ sung từ IMDb thông qua OMDB API: Để khai thác tốt hơn khả năng hiểu ngữ nghĩa của mô hình LLM và phục vụ xây dựng hệ thống web, nhóm đã thu thập thêm các trường dữ liệu từ IMDb bao gồm:

- Tên đạo diễn
- Diễn viên
- Tóm tắt cốt truyện
- Ngôn ngữ
- Thời lượng
- Poster
- Ngôn ngữ
- Độ dài

6.2 Xử lý dữ liệu

Đối với dữ liệu phim, nhóm đã kết hợp với dữ liệu về tags tự do, tuy nhiên ko phải phim nào cũng được gán tag nên kết quả cuối cùng thu lại được hơn 10000 phim có đầy đủ tag. Kết hợp với những dữ liệu lấy thêm từ IMDB.

Đối với dữ liệu là đánh giá của user, nhóm chỉ lấy ra những user đã xem ít nhất 100 phim. Sau đó sắp xếp phim đã xem của từng user theo timestamp mà họ đã xem để phục vụ cho việc chia train-test trong quá trình đánh giá.

6.3 Đánh giá

Quá trình đánh giá được thực hiện trên 115 user đã xem ít nhất 100 phim

Ý tưởng đánh giá: Mỗi user sẽ có lịch sử xem phim riêng và mỗi user đều xem phim khác nhau nên phần chia train test sẽ được thực hiện với mỗi user, với tỉ lệ 75/25. Do các phim của mỗi user đã được sắp xếp dựa trên thời gian họ xem nên đảm bảo được tính dự đoán trong tương lai của hệ thống RS.

6.3.1 Content-based Filtering (LLM Embedding)

Metric	K=10	K=20	K=50
Precision	6%	6%	6%
Recall	1%	2%	5%
NDCG	5%	5%	5%
HR	46%	63%	89%

Nhận xét:

- Chưa khai thác hết dữ liệu người dùng.
- Chỉ sử dụng khả năng embed của LLM.

- Đánh giá còn thấp.

6.3.2 Hybrid Filtering

Metric	K=10	K=20	K=50
Precision	0.1330	0.1113	0.0864
Recall	0.0226	0.0378	0.0701
NDCG	0.1463	0.1265	0.1110
HR	0.6087	0.7043	0.8174

Hệ thống gợi ý đạt hiệu suất tốt ở các mức độ khác nhau của K. Precision và NDCG cao ở K nhỏ → hệ thống đưa ra những đề xuất chất lượng ở vị trí đầu. Recall và HR cao ở K lớn → hệ thống có khả năng bao phủ nhiều item phù hợp hơn khi mở rộng danh sách đề xuất.

6.3.3 So sánh

	HR@10	HR@20	Precision@10	Precision@20
LE	46.2	63.1	6.0	6.0
CF + TS	43.5	57.4	6.3	6.0
CF + TS + MS	60.9	70.4	13.1	11.1

Chú thích:

- **LE:** Phương pháp LLM Embedding cho Content-base (Cách 1)
- **CF + TS:** CF kết hợp với TS với tỷ lệ 70/30 (Cách 2)
- **CF + TS + MS:** CF + TS kết hợp thêm Movies Score

Kết quả cho thấy mô hình Hybrid đạt hiệu suất vượt trội so với phương pháp gợi ý truyền thống. Đặc biệt, khi so sánh giữa hai công thức tính Hybrid Score – một công thức có kết hợp thêm Movie Score và một công thức chỉ bao gồm CF Score và Semantic Score, kết quả cho thấy việc đưa thêm yếu tố Movie Score đã cải thiện rõ rệt chất lượng gợi ý. Điều này chứng tỏ chất lượng và độ nổi tiếng một phim ảnh hưởng rất nhiều đến bộ phim đấy có được yêu thích bởi người xem hay không ngoài độ phù hợp với sở thích của họ.

7 Ứng dụng

Source code:

<https://github.com/anbit192/LLM-rcm>

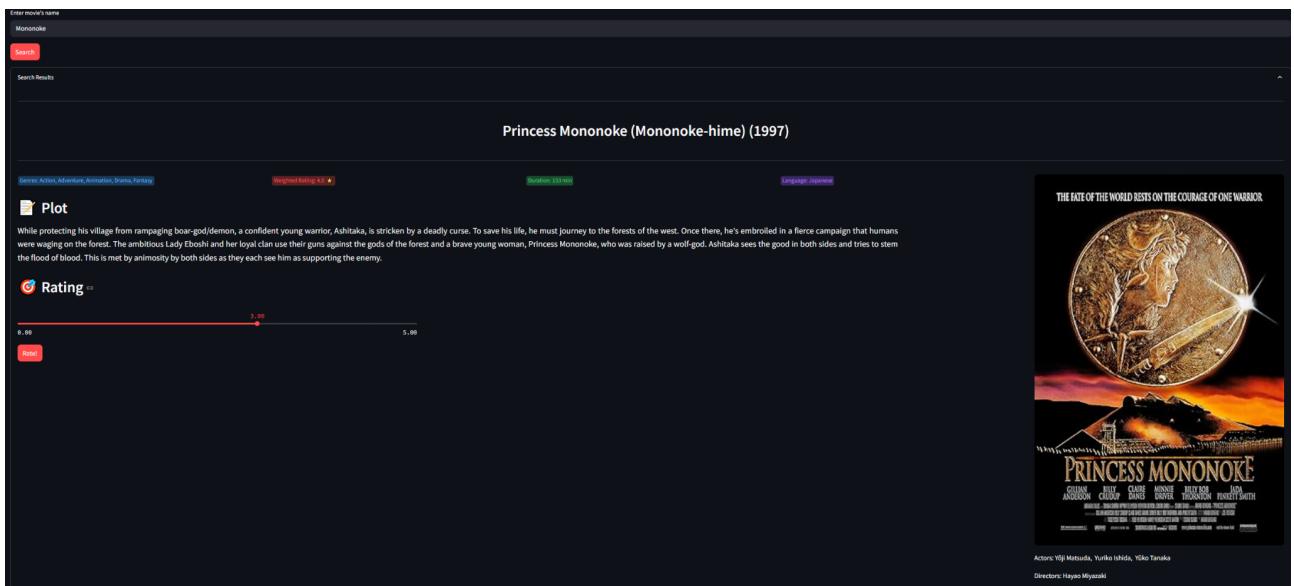
Folder drive:

<https://drive.google.com/drive/folders/1yid0rPVfn37BtR5h1HJ-9FbReJ-BLt6S?usp=sharing>

7.1 User Interface

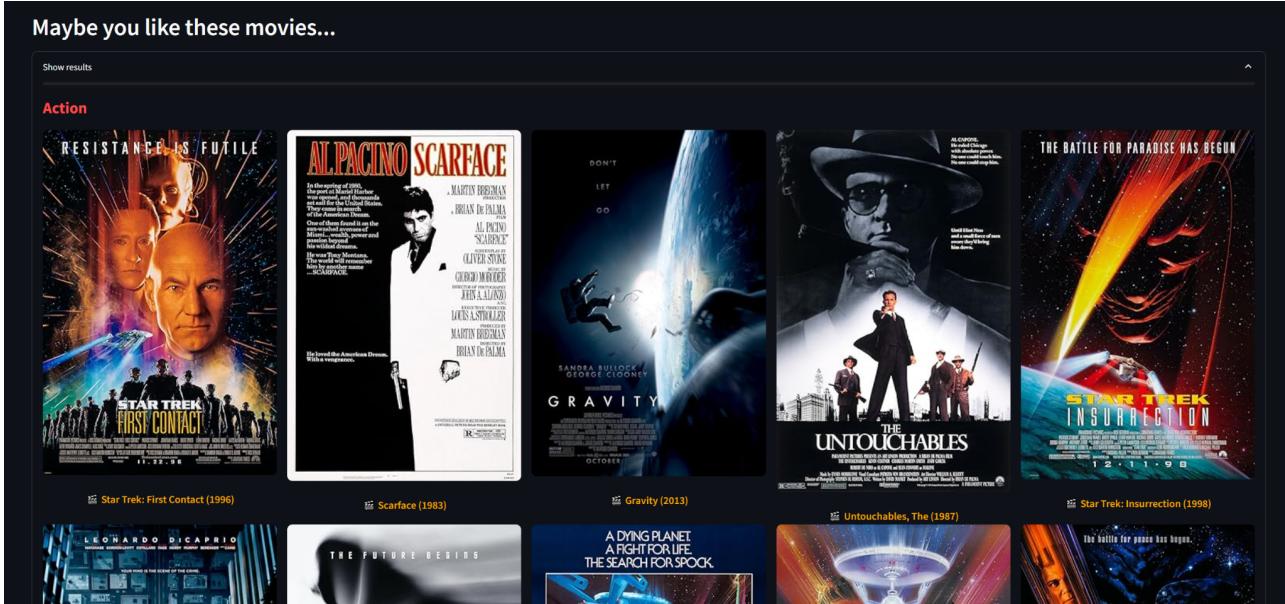
Xây dựng giao diện bằng Streamlit với các thành phần:

- **Giao diện login:** Cho phép đăng nhập người dùng.
- **Giao diện chính:**
 - Hộp search: Tìm kiếm và hiển thị thông tin phim cũng như đánh giá.



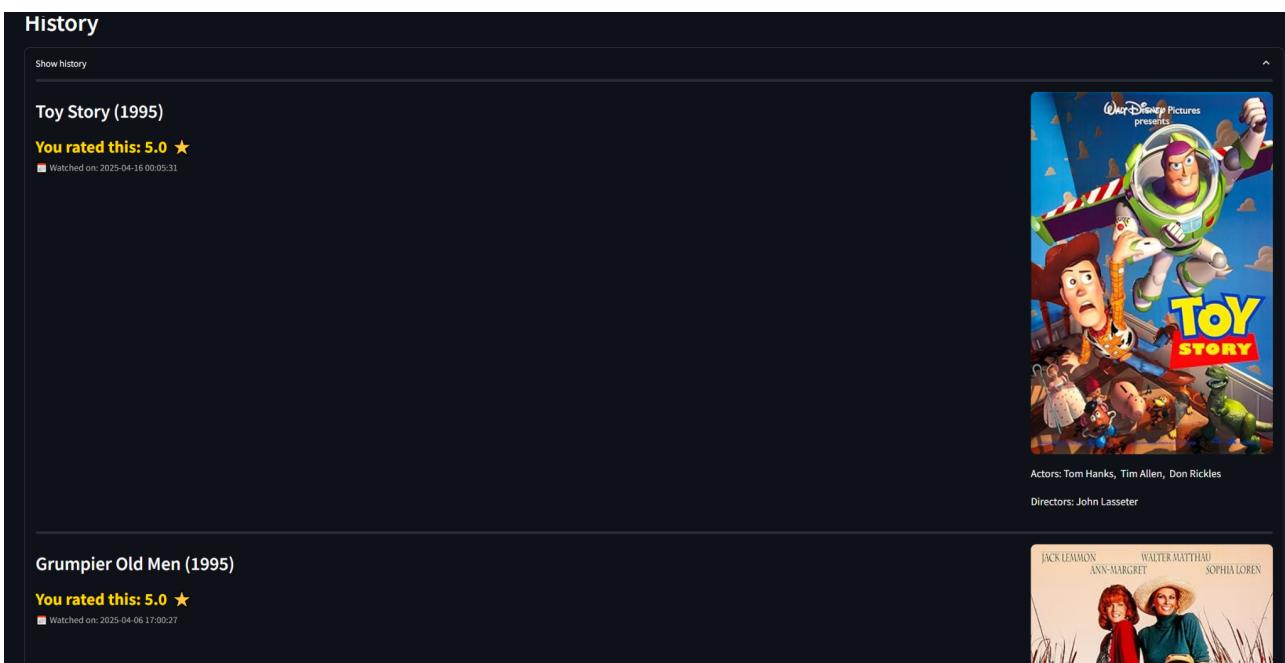
Hình 6: Giao diện tìm kiếm

- Hộp recommend: Hiển thị các gợi ý phim.



Hình 7: Giao diện chính

- Hộp lịch sử xem: Hiển thị lịch sử xem của người dùng.



Hình 8: Giao diện lịch sử

7.2 Chức năng

- Đăng nhập người dùng.
- Tìm kiếm phim
- Gợi ý phim dựa trên lịch sử xem.
- Yêu cầu đăng nhập trước khi cho phép đánh giá.

- Nếu người dùng chưa có lịch sử xem, hiển thị phim gợi ý mặc định.
- Lưu trữ và hiển thị đánh giá vào cơ sở dữ liệu.

8 Kết luận

Trong phạm vi dự án này, nhóm đã xây dựng thành công một hệ thống gợi ý phim sử dụng phương pháp **Content-based (LLM Embedding)** và **Hybrid Filtering**, kết hợp giữa **Collaborative Filtering**, **Summary Text**, và yếu tố **Movie Score** (phản ánh độ phổ biến và độ tin cậy của phim). Nhóm cũng đã xây dựng một **web app** phục vụ cho việc lưu trữ và gợi ý phim cho người dùng.

Kết quả đánh giá cho thấy mô hình **Hybrid** mang lại hiệu quả gợi ý vượt trội so với các phương pháp đơn lẻ. Điều này khẳng định vai trò quan trọng của việc tích hợp nhiều nguồn thông tin trong việc nâng cao độ chính xác và mức độ phù hợp của hệ thống gợi ý.

Tổng thể, hệ thống không chỉ tận dụng được khả năng học sở thích từ dữ liệu người dùng, mà còn hiểu được ý nghĩa nội dung phim ở mức sâu thông qua mô hình **LLM**, đồng thời bổ sung yếu tố đánh giá khách quan từ cộng đồng. Mô hình này có khả năng mở rộng tốt và phù hợp với các bài toán gợi ý trong thực tế.

Trong tương lai, nhóm sẽ cải thiện thêm hệ thống bằng cách thử nghiệm với những LLM khác và chất lượng hơn. Prompt cũng có thể được tối ưu để trích xuất thêm đặc trưng hữu ích hơn cho việc gợi ý. Web app có thể được mở rộng để phục vụ cho nhiều người dùng hơn sau này.

Tài liệu

- [1] Liu, Q., Zhao, X., Wang, Y., et al. (2025). Large Language Model Enhanced Recommender Systems: A Survey.
- [2] Wu, L., Zheng, Z., Qiu, Z., et al. (2024). A Survey on Large Language Models for Recommendation.
- [3] Wang, Q., Li, J., Wang, S., et al. (2024). Towards Next-Generation LLM-based Recommender Systems: A Survey and Beyond.